REMARKS

Claims 1-36 are pending in the application. Claims 1-36 stand rejected. Claims 1-31 stand rejected. Claims 1, 7, 8, 11, 26, 27, and 32 have been amended herein. Applicant traverses the rejections.

Claim Rejections - 35 U.S.C. § 112

Claims 1-25, 27, and 28 stand rejected under 35 U.S.C. 112, second paragraph. Claims 1, 7, 8, 11, and 27 have been amended to address the instant rejection. Because claims 1, 8, 11, and 27 have been amended to address antecedent basis, these amendments are not narrowing amendments. Although applicant believes that the term "substantially" as used in claim 7 is a definite term, applicant has amended claim 7 to remove the term in order to expedite prosecution of the application. Because of the amendments to claims 1, 7, 8, 11, and 27, the rejection of the dependent claims of these claims should also be removed.

Claim Rejections - 35 U.S.C. § 102

Claims 1-36 stand rejected under 35 U.S.C. § 102(b) as being anticipated by Johnson (XML JavaBeans series, Part 1-3, published 2-7/1999, JavaWorld; hereinafter "Johnson"). Applicant traverses the instant rejection.

Claim 1 recites that private state data is stored for an object that has been created within an object development environment. In the manner suggested on page 19 of the office action, claim 1 has been amended to better distinguish between private state data and public state data, such that "private state data is an object's internal state data," and

"public state data is data that can be retrieved via public method calls or public fields." As recited in claim 1, the private state data is obtained by querying an object with respect to its private object state. In this way, the method of claim 1 can allow for storing both private data as well as public data for later restoring. Querying private state data can be performed, such as for example in the following manner:

> Using introspection, a JavaBean's properties, methods and events can be exposed automatically by a builder tool if the programmer follows the proper JavaBean design patterns. For JavaBeans that do not follow the JavaBean design patters [sic] or for JavaBeans in which the programmer wants to customize the exposed set of properties, methods and events, the programmer can supply a BeanInfo class that describes to the builder tool how to present the features of the bean.
> (See, Deitel & Deitel, *Java, How To Program*, page 1223, third edition, 1999; which passage is attached hereto.)

(e.g., see FIGS. 3A and 3B in applicant's specification wherein flow diagrams show an example of objects that are used to capture the state information of frame objects.)

In contrast, the Johnson reference discusses saving public state information and not private state data as required in claim 1. As clarified in claim 1, public state data is significantly different than private state data. Public state data is data that can be retrieved via such methods as public method calls or public fields. Unless made accessible as described above, private state data is an object's internal state data which is not accessible via the object's public interface. Since many objects contain internal states that should be captured in order to properly restore them later (such as at runtime), the method of claim 1 allows for the capturing and restoring of this important state data, whereas Johnson cannot.

The office action maintains on page 21 that the "exposure of the private data directly violates encapsulation and leads to harm the public interest where the sensitive

data can be read by anyone (or even modified)." Applicant respectfully disagrees. Claim 1 recites that the persisting of private object state data is performed within an object development environment, such as by the software component designer. The persistence of object state data in a human-understandable format (such as a text human-readable format) allows a component designer to modify object structures (such as object classes) outside of the development environment through a wide range of techniques. For example, the component designer may directly edit the file to alter the object structure or use different file editors (e.g., XML parsing mechanisms) to alter the object structure. The component designer can more easily correct errors or update object structures without having to reenter the development environment. Accordingly, the method of claim 1 does not harm the public interest, but instead has many beneficial uses. Because the rejection of claim 1 is traversed, claim 1 is allowable and should proceed to issuance. Because claim 1 is allowable, the dependent claims of claim 1 are also allowable and should proceed to issuance.

Claim 26 has been amended to better distinguish between private state data and public state data, and because Johnson discusses saving public state information and not private state data as required in claim 26, claim 26 is allowable and should proceed to issuance. Because claim 26 is allowable, the dependent claims of claim 26 are also allowable and should proceed to issuance.

Applicant disagrees with other positions in the office action. For example, the office action maintains for claim 11 on page 22: "Johnson discloses a XML formatter. XML document is human readable, and therefore the object structure can be easily manipulated. Therefore, the XML formatter in Johnson allows the storing of object state

data with a specific restoration order." Applicant respectfully disagrees. Claim 11 first determines that first private object state data is to be restored before second private object state data because of an interdependency between the first private object state data and the second private object state data"; and then generates "the computer-readable file such that a restoration order is provided in the computer-readable file." There is no disclosure in Johnson whatsoever of determining a restoration order "because of an ***interdependency*** between the first private object state data and the second private object state data" as required by claim 11. Moreover, the general recital by the office action that the object structure could be manipulated in such a manner is a bald assertion and therefore is improper -- it is improper to determine a claim invalid using a "bald assertion that 'substitution of one type of [approach] for another in the system of [the prior art] would have been within the skill of the art,' [. . . without offering] any support for or explanation of this conclusion." In re Fine, 837 F.2d 1071, 1074 (Fed. Cir. 1988). Because Johnson does not teach, disclose or suggest the limitations of claim 11, claim 11 is allowable.

With respect to claim 32, claim 32 expressly recites in combination with its other limitations a "means for determining that an interdependency exists between first private object state data and second private object state data" for use in specifying a restoration order. Because the Johnson reference does not disclose, teach or suggest any type of determination of an interdependency existing between first and second private object state data, claim 32 is allowable and should proceed to issuance. Because claim 32 is allowable, the dependent claims of claim 32 are also allowable and should proceed to issuance.

CONCLUSIONS

For the above-stated reasons, claims 1-36 are allowable over the cited reference. Therefore, the examiner is respectfully requested to pass this case to issuance.

Respectfully submitted,

Date: March 2, 2005

By: _____

John V. Biernacki
Reg. No. 40,511
JONES DAY
North Point
901 Lakeside Avenue
Cleveland, Ohio 44114
(216) 586-3939

JAVA™ HOW TO PROGRAM

THIRD EDITION

H. M. Deitel
Deitel & Associates, Inc.

P. J. Deitel
Deitel & Associates, Inc.

Associates,

isit the Pren-

the last pages

PRENTICE HALL, Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Deitel, Harvey M.

Java : how to program / H.M. Deitel, P.J. Deitel. -- 3rd ed.

p. cm. -- (How to program series)

Includes bibliographical references.

ISBN 0-13-012507-5

1. Java (Computer program language) I. Deitel, Paul J.
II. Title. III. Series.

QA76.73.J38D45 1999 99-37546

005.13'3--dc21 CIP

Acquisitions Editor: ***Petra J. Recter***
Production Editor: ***Camille Trentacoste***
Chapter Opener and Cover Designer: ***Tamara Newnam Cavallo***
Buyer: ***Pat Brown***
Editorial Assistant: ***Sarah Burrows***

A Pearson Education Company
Upper Saddle River, New Jersey 07458

Printed in the United States of America

10 9 8 7 6 5 4 3

ISBN 0-13-012507-5

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Prentice-Hall Singapore Pte. Ltd., Singapore
Editora Prentice-Hall do Brasil, Ltda., Rio de Janeiro

TO

Dolo

Know
publis

You a
work

Thank

Love,

Barba

- There are two ways to load the bean into the **BeanBox**—place the JAR file in the `BDK1.1\jars` directory or use the **File** menu's **LoadJar...** option to locate the JAR file on your system and load it into the **BeanBox**'s **ToolBox**. If you place the JAR file in the `BDK1.1\jars` directory, it will automatically be loaded into the **ToolBox** when the **BeanBox** is executed.
- A read-write property of a bean is defined as a pair of *set/get* methods of the following form:

```
public void set PropertyName( DataType value )
public DataType get PropertyName()
```

  If the property is `boolean` data type, the *set/get* method pair is normally defined as

```
public void set PropertyName( boolean value )
public boolean is PropertyName()
```

  When a builder tool examines a bean, if it locates a *set/get* method pair with the preceding formats, the builder tool exposes that pair of methods as a property in the bean.
- A bound property causes the object that owns the property to notify other objects that there has been a change in the value of that property. All registered `PropertyChangeListener`s are automatically notified when the property's value changes. The `java.beans` package provides interface `PropertyChangeListener` to create listeners that can receive notification of the property change, class `PropertyChangeEvent` to provide information to a `PropertyChangeListener` about the change in the property's value and class `PropertyChangeSupport` to provide the listener registration and notification services.
- To support registration of listeners for changes to a bound property, a bean must provide methods `addPropertyChangeListener` and `removePropertyChangeListener`. Each of these methods typically calls the corresponding method in a `PropertyChangeSupport` object that provides the event notification services when the property value changes.
- When a bound property changes, the registered `PropertyChangeListener`s must be notified of the change. Each listener is presented with old and new property values when notified of the change. The `PropertyChangeSupport` object's `firePropertyChange` method notifies each registered `PropertyChangeListener`.
- When a bean provides bound properties, there will be a **Bind property...** menu item in the **BeanBox**'s **Edit** menu. Select the **Bind property...** menu item from the **Edit** menu to display the **PropertyNameDialog**. Select the bound property. Connect the red target selector line to the target of the property change event to select the method to call.
- Using introspection, a JavaBean's properties, methods and events can be exposed automatically by a builder tool if the programmer follows the proper JavaBean design patterns. For JavaBeans that do not follow the JavaBean design patters or for JavaBeans in which the programmer wants to customize the exposed set of properties, methods and events, the programmer can supply a `BeanInfo` class that describes to the builder tool how to present the features of the bean.
- Every `BeanInfo` class must implement the `BeanInfo` interface that describes the methods used by a builder tool to determine the features of a bean. Class `SimpleBeanInfo` provides a default implementation of each method in the `BeanInfo` interface. The programmer can extend this class and selectively override methods of this class to implement a proper `BeanInfo` class.
- Override method `getPropertyDescriptors` to return an array of `PropertyDescriptor` objects in which `PropertyDescriptor` indicates a property a builder tool should expose.
- If the *set/get* methods for a property do not use the JavaBeans naming convention for properties, there are two other `PropertyDescriptor` constructors in which the actual method names are passed. This allows the builder tools to use nonstandard property methods to expose a property for manipulation by the programmer at design time.